

**Graduate Research Paper-IT485**  
**Fall-2015**  
**Author: Nikunj Ratnaparkhi**

**"DYNAMICALLY PROGRAMMABLE" CSS-Transforming the world of  
Front-End Web Development**

**Contents**

I have written about the topic "DYNAMICALLY PROGRAMMABLE" CSS-Transforming the world of Front-End Web Development as a graduate research in IT485 in Fall-2015. In this report, I have covered below mentioned points:

- I have included some executable code snippets to prove my findings about how can we make CSS dynamically programmable.
- I have illustrated about why it's not a good practice to make changes in local-css or localized styling if we can make our css programmable based on the changing needs of front-end development.
- I have discussed the extent to which we can implement CSS programming
- I have provided my findings about the technical limitations of css pre-processors.
- I have discussed how css pre-processors such as Less, Sass and Stylus has evolved from scratch. For example, how Less has evolved from 2009 to the most stable release in 2014.
- What are the short-comings of these frameworks in the production environment and real business world?
- How a heavy CSS and styling can impact energy level at hardware like battery and speed of processing in desktop and mobile applications.
- Things which programmable CSS can invent or invented in the recent years.
- Recent discoveries and their practical impact in the world of web development.
- Conclusion
- References to the report.

## **Abstract**

In the area of front-end web development CSS has played a big role in maintaining the aesthetic beauty of modern web application. With the advent of responsiveness and need to develop cross-platform web applications, CSS has provided great benefit to make the life of programmers easy while making end user addicted to the websites. However, after years of development programmers have realized that CSS lacks flexibility. Though it works fine for small sites, it starts breaking down as the complexity of web applications increases. Consider an example, if we want a myriad of color-combinations in our website then it's a bad practice (and very time consuming) to figure-out hex code of each color and make changes in the code. How about if we are able to lighten or darken the css-colors programmatically? This example is just a tip of Ice-Berge as compared to what we have discussed in this paper.

This paper discusses the idea of introducing dynamically-programmable CSS from very simple example and progresses toward the sophistication i.e. impact on hardware like batteries in mobile devices when heavy CSS is plugged into a mobile applications. Also, with the detailed coverage of all the points mentioned above, this paper discusses about css pre-processors like Less, Sass, Stylus and discusses quality of one over another.

## **Report**

In the rapidly growing world of e-commerce, internet applications have played a crucial role in uplifting outcome of electronic marketing and commerce. All the market players strive to provide best possible services to their clients. However, in the era where people have numerous options, they will always be attracted to the e-commerce application which provides most awesome look, feel and least response time.

CSS preprocessor make your CSS file super charged by instilling in them an ability to define (and use) variables, functions, reusable chunks ("Mixins") and other numerous features. Imaging a situation where we are given a task to develop/maintain a sophisticated web application which has n number of css files and each css file have some thousand lines of code. Also, we need to make sure that the website runs successfully on all the browsers and mobile devices. In such complex projects, if we use plain CSS then we need to make large number of customization and include different versions of CSS files with respect to each browser and each device. There is a better (way too better) trick to handle such complex scenarios.

If we go back and analyze more about css, we can easily say that css is primitive and incomplete in many senses. General programming concepts like building a function, reusing variables, inheritance etc. are few among many shortcomings of css. With years of evolution of internet applications, we have set some standards to make best use of css by separating definition in separate smaller files and importing them into the main file. However, these approach of standardization does not solve the problem of code repetition and maintainability. Another approach was to make object oriented css. In this case applying two or more class definition to an element provides an ability to acquire styles by element but does not solve the problem of maintainability. CSS3 preprocessor is a language which provides us an ability to add cool, inventive features to our CSS without worrying us about cross-browser compatibility. With their advanced features they help to write reusable, maintainable and extensive codes in css. We can easily increase our productivity and decrease the amount of code we write in css files. The code written in CSS3 preprocessor is compiled to normal css –which can be further understood by different kind of browsers. So, it eliminates an issue of browser compatibility and code redundancy when we think about cross-browser compatible application.

There are 3 types of popular css3 preprocessors- Sass, Less and Stylus - which makes the css dynamically programmable and provides an ability to change the behavior (look & feel) based on the target browser.

Now, consider a situation where we have very complex web application and we want to change the color of all the buttons to green. If we use normal CSS, we will go and make changes in all the css corresponding to the button in all the pages or the pages which are linking the css. Later, if our client does not like the color and asks us to change the button to some other color, it will be as time consuming as it was in the previous change. So, CSS3 pre-processors is the better and quick solution.

Sass and LESS both uses standard css syntax. Sass uses .scss extension while LESS used .less extension. For example, the below code works perfectly for both Sass and LESS.

```
/* test.scss or test.less */  
h1 {  
  color: green;  
}
```

**Fig-1**

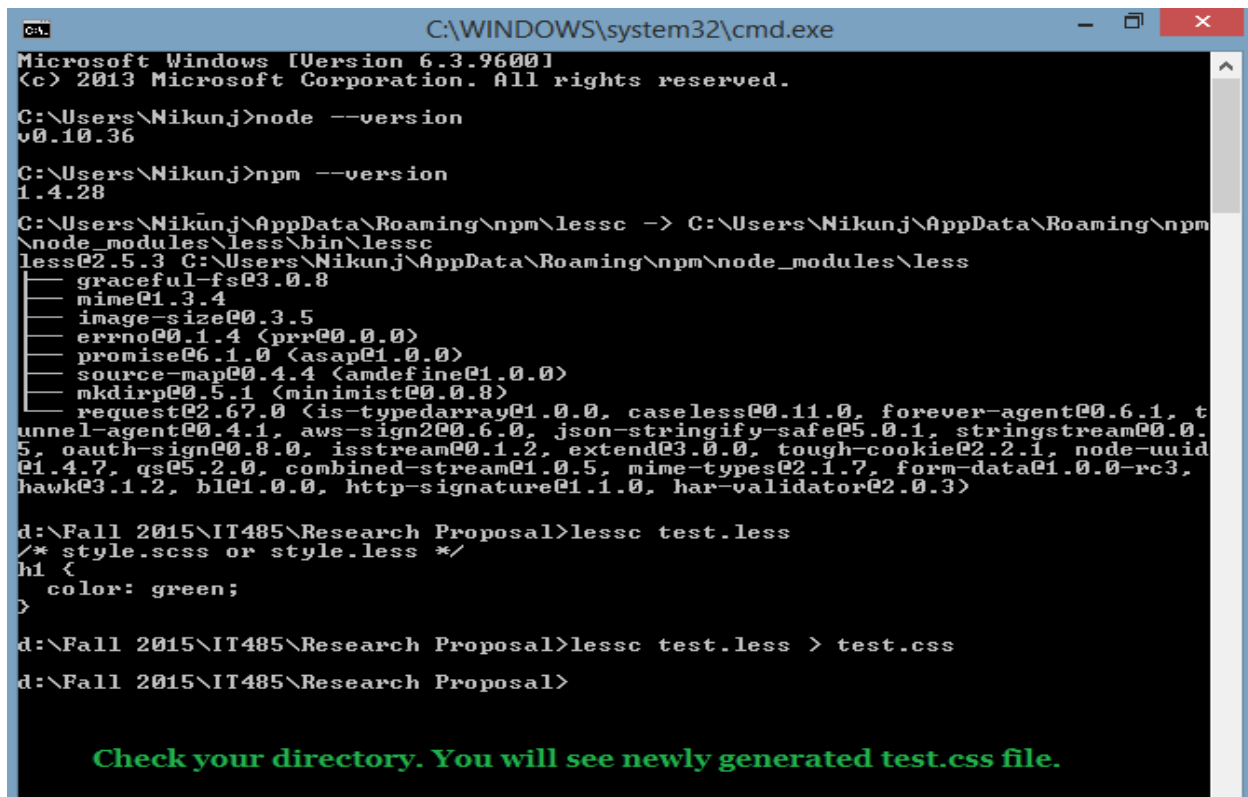
Stylus, scss and less varies in syntax but from functionality-wise all there are similar as far as the scope of this research paper is concerned, we will use less for ease of understanding.

So, if we save the code mentioned in the Fig-1 by .less extension, Less compiler – which is written in JavaScript- will convert the Less code into CSS, which can be understood by client browser. This less to css conversion can happen in 3 ways: first Just in time, second by using lessc compiler through command line, and third with some apps like CodeKit or LiveReload.

Just in time compilation of .less files to .css file in production is very inefficient way of generating css for the websites. Because, less.js makes Ajax calls to grab .less files and inject resulting css into the head of the document. Since, it is extremely inefficient in terms of performance, using conversion through command line or using an application is a better way.

Below is the steps to use lessc compiler through command line:

1. Install Node.js on your development machine.
2. Run: `npm install less -g` command
3. Then run the below command to compile your less file to css file:
  - `lessc test.less > test.css`



```
C:\WINDOWS\system32\cmd.exe
Microsoft Windows [Version 6.3.9600]
(c) 2013 Microsoft Corporation. All rights reserved.

C:\Users\Nikunj>node --version
v0.10.36

C:\Users\Nikunj>npm --version
1.4.28

C:\Users\Nikunj\AppData\Roaming\npm\lessc -> C:\Users\Nikunj\AppData\Roaming\npm\node_modules\less\bin\lessc
less@2.5.3 C:\Users\Nikunj\AppData\Roaming\npm\node_modules\less
├── graceful-fs@3.0.8
├── mime@1.3.4
├── image-size@0.3.5
├── errno@0.1.4 <pr@0.0.0>
├── promise@6.1.0 <asap@1.0.0>
├── source-map@0.4.4 <amdefine@1.0.0>
├── mkdirp@0.5.1 <minimist@0.0.8>
├── request@2.67.0 <is-typedarray@1.0.0, caseless@0.11.0, forever-agent@0.6.1, tunnel-agent@0.4.1, aws-sign2@0.6.0, json-stringify-safe@5.0.1, stringstream@0.0.5, oauth-sign@0.8.0, isstream@0.1.2, extend@3.0.0, tough-cookie@2.2.1, node-uuid@1.4.7, qs@5.2.0, combined-stream@1.0.5, mime-types@2.1.7, form-data@1.0.0-rc3, hawk@3.1.2, bl@1.0.0, http-signature@1.1.0, har-validator@2.0.3>
└── unnel-agent@0.4.1, aws-sign2@0.6.0, json-stringify-safe@5.0.1, stringstream@0.0.5, oauth-sign@0.8.0, isstream@0.1.2, extend@3.0.0, tough-cookie@2.2.1, node-uuid@1.4.7, qs@5.2.0, combined-stream@1.0.5, mime-types@2.1.7, form-data@1.0.0-rc3, hawk@3.1.2, bl@1.0.0, http-signature@1.1.0, har-validator@2.0.3>

d:\Fall 2015\IT485\Research Proposal>lessc test.less
/* style.scss or style.less */
hl {
  color: green;
}

d:\Fall 2015\IT485\Research Proposal>lessc test.less > test.css

d:\Fall 2015\IT485\Research Proposal>
```

**Check your directory. You will see newly generated test.css file.**

**Fig-2**

And as mentioned, there is a third approach, there are numerous application which can get the job done for you by converting your .less file to .css on the go.

Let's, consider the below html file as an example to understand this magical conversion easily.

```
<!doctype html>
<head>
  <link rel="stylesheet" href="styles.css">
</head>
<body>
  <div class="shape" id="shape1"></div>
  <div class="shape" id="shape2"></div>
  <div class="shape" id="shape3"></div>
</body>
</html>
```

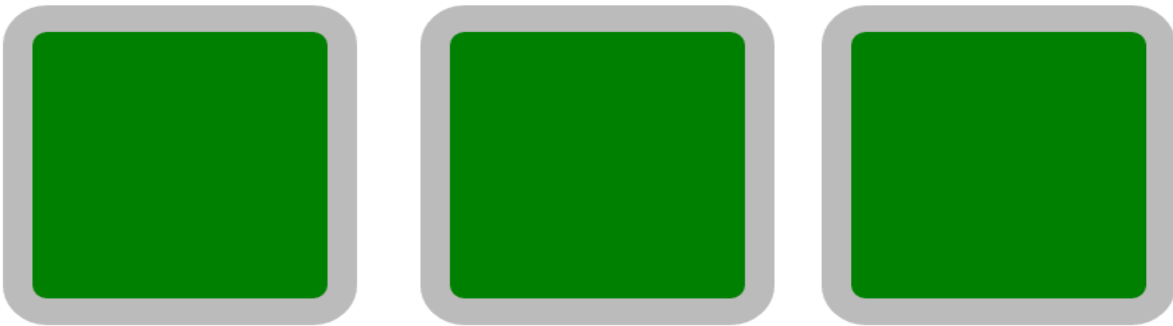
**Fig-3**

And, save the below code with (let's say) Style.less name in the same directory.

```
.shape {
  display: inline-block;
  width: 200px;
  height: 200px;
  background: green;
  margin: 20px;
  border: 20px solid #bbbbbb;
}
```

**Fig-4**

Now run the command `lessc test.less > test.css` and you can see a .css file is generated in your folder. Now, open the html page in your web browser. The page will look like this:



**Fig-4**

Now, this looks like a usual css (nothing new!). So, to make use of dynamic properties and all the benefits of css3 pre-processors, make the below modifications in the above code.

- Declare the less variable. Less variable can be declared using @ symbol. We can give them a name and value and then we can use them anywhere we want.
- For example, in the code (Fig-5), we declared a variable @defaultThemeColor – which holds the value of another variable @lightGreen.
- In your complex site, if you are asked to make changes in the shade of green color, you just need to make changes at one place and your less compiler will update your css file on the go (depending on the conversion method you use).

```
@lightGreen: #dfd;

@defaultThemeColor: @lightGreen ;

.shape{
  display:inline-block;
  width:200px;
  height:200px;
  background:@defaultThemeColor;
  margin:20px;|
  border:@borderSize solid @borderColor;
}
```

**Fig-5**

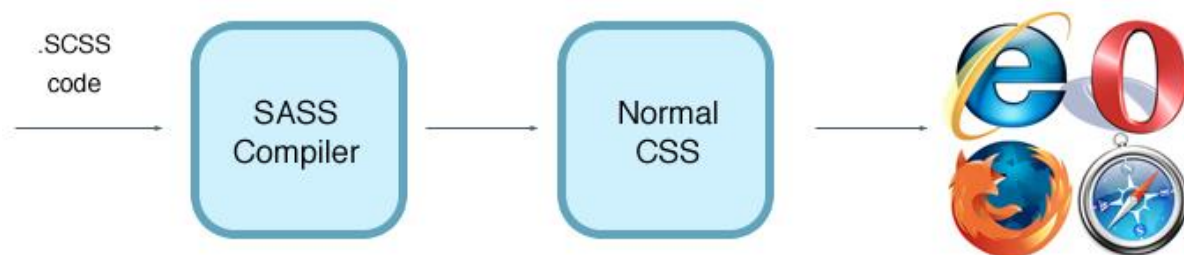
- Now, the compile the above .less file in the same approach to convert into .css file.
- If our client wants to change the border color to be lighter than the default theme color then we do not need to go and search for the hex code of the border color and copy and replace every time a change is demanded by the client. We can simply use below lines of code to lighten or darken color. Apart from these, there are numerous other pre-defined functions in css3 pre-processors which we can make use of to save the time and increase the efficiency of programmers.

```
$borderColor:$defaultThemeColor - #222;

$borderColor:darken(desaturate($defaultThemeColor, 100%), 20%);
```

**Fig-6**

So, in a nut-shell, below is the approach which is getting followed to convert LESS, SCSS or STYLUS files to normal css.



**Fig-7** Reference [7]

There are various reasons why css3 pre-processors has made a dominance over the cliché syntax of local css in internet application industry. CSS pre-processors allows additional leverage over css by providing non-verbose syntax. For example, css3 pre-processor has:

- Nested Syntax
- Ability to define, use and call variables
- Ability to define, use and call mixins
- Apply Mathematical functions to for example, lighten/darken color themes.
- Joining of multiple files

Apparently, we cannot achieve these robust functionality using normal CSS. And, after analyzing these robust features, css no more looks like plain, dead style sheets. Apart from the above benefits, there are several other benefits of using css pre-processors over localized css. Such as:

- CSS pre-processors make our CSS DRY (Don't Repeat Yourself) and regardless to say, it saves a lot of time.
- It makes our CSS more easy to organized
- Setting up preprocessor files for a project is surprisingly easy. With Less, Sass or Stylus, a simple command line command will instruct the compiler to watch your files, recompiling your CSS automatically every time you save your changes.
- We can use some of the spicy CSS3 properties to make our site look more neat, prettier and polished.
- It is quick and easy to learn and pick-up.
- There are some frameworks built on top of CSS preprocessing languages which do even more heavy lifting. For example, Compass- a framework built on top of Sass, automatically generates vendor-specific CSS3 properties, has lots of useful functions for generating grids, sticky footers, to name a few.
- One of the core attributes of a good CSS code is reusability. Using CSS pre-processor, we can identifying reusable patterns across the UI, and create smart chunks of reusable rules. LESS supports this in a big way, through "mixins". Mixins are reusable sets of styles which can be used throughout the stylesheet.
- Extend is one of the most beautiful and interesting feature of SASS. This is similar to mixin, but brings a ton of other goodies with it as compared to mixin. Using "@extend" enables a selector to inherit the rules of another selector.
- *SASS comes loaded with much more features like if, for loops, @debug, @warn, @while etc., which adds a lot of value and productivity to the team, as a whole and makes the stylesheet file modular, scalable and maintainable.*<sup>7</sup>



On the other hand, there are various technical and feasibility limitations of using CSS pre-processors. Also, if we talk about very sophisticated project in the industry, then the very first thing which any new technology has to suffer with is the acceptance from the users. In Dec-2014, the famous envato.com conducted a polling regarding the acceptance of pre-processors. Based on the results of below polling, we can conclude that any new technology has to bear the brunt of acceptance in the early years.

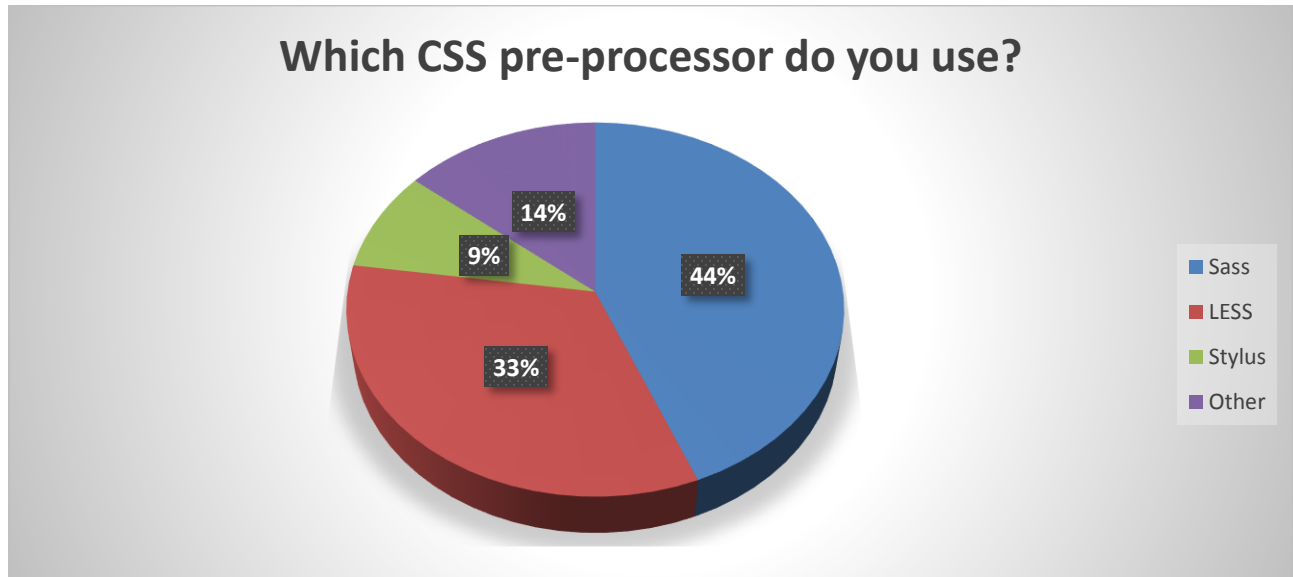


Fig-8

There are other technical limitations when we try to inject css3 pre-processors in the complex projects in the production environment:

- It doesn't work without compiling so, there is no ability to live edit a site for quick changes. In most cases, we cannot always make changes in the pre-processor code and update it via FTP, because in the real world – with deadlines and lot of standards to follow – it sometimes consumes more time. Therefore, for the same reason, any changes you might need to make to the CSS in the future will have a few extra steps, too.
- CSS pre-processor has nested nature. So, in the wrong hands, it can make CSS very ugly.
- It is very easy to make mistake at one place in the pre-processor file and then it will make the changes in the css at multiple place. Which therefore make look and feel worst.
- Sometimes debugging is very tedious when we use css pre-processors. Due to having a compilation steps, sometimes the browser does not interpreting the source files, meaning the CSS line numbers becomes irrelevant when trying to debug.

- The websites which has target users who access the website from the older browsers - debugging can become as hard as programming for those website if we use css pre-processors. Because, due to unavailability of supporting files it becomes hard to trace down the source of problem.
- CSS pre-processor requires extra tooling for real-time compilation or make use of dynamic css3 properties in the website. Some industries do not support tools because of security constraints and this is the point where it becomes hard to make use of awesomeness of css3 pre-processors.
- Since the css pre-processor is lot more compressed as compared to the .css file it may lead to performance issues as the developers will not be aware about final size of their css. Source file size can be deceiving because the generated CSS is likely to be significantly larger than the source file, potentially causing performance problems. Ultimately, what is generated is out of your control.
- Even after following all the rules of CSS pre-processors, maintainability can sometimes be compromised. As an example, I have seen @lightGreen (instead of #00FF00) across various CSS files. And if the color change is for something that isn't a @lightGreen, then we have to do a search and replace anyway - which is not recommended as per the best practice which comes along with css pre-processor's use.

Until now, we learnt about css, css3 and the power css3 pre-processors which makes our UI more polished, clean and awesome. However, thing were not as easy in the past as it looks today. Here, we will take a deeper look at the evolution of CSS3 pre-processor from the days when programmers used to write css from scratch and test the look and feel by reloading entire page every time they make even the slightest change in the css.

Programmers used to manage css by hand and as the size of project increased, it became a nightmare to handle intricate css and make css consistent across all the pages in the browser. To overcome this issue, css pre-processor came to the rescue. They made the life of front-end developer very easy and manageable. However, because of lot of limitations of css pre-processors they does not seem like a plausible and robust solution in the rapidly changing world of internet application development.

Mobile devices and access of website from the mobile devices has gained a lot of popularity in the recent years. In the research paper published in ACM Digital library "Who Killed My Battery: Analyzing Mobile Browser Energy

Consumption”, authors have demonstrated their findings about dominant role of css in consumption of battery of mobile phone when the website is access from the mobile browser. In the article author mentioned that even though the popularity of mobile browsers and application has gained over the years, the energy consumed by a phone browser while surfing the web is poorly understood. They presented an infrastructure for measuring the precise energy used by a mobile browser to render web pages. They, then measure the energy needed to render financial, e-commerce, email, blogging, news and social networking sites. Their tools are sufficiently precise to measure the energy needed to render individual web elements, such as cascade style sheets (CSS), JavaScript, images, and plug-in objects. Their results clearly shows that for popular sites, downloading and parsing cascade style sheets and JavaScript consumes a significant fraction of the total energy needed to render the page.<sup>8</sup> they also made changes in some popular sites by removing the heavily loaded CSS and accessed the same site under the similar environment. And, they discovered that the site consumes 30% less energy than the previous situation. So, css heavy css and css pre-processor not only a concern for the web developer but also has various issues with the performance at hardware level. So, what next?

In the recent article published by Lyza Danger Gardner – author of Head First Mobile Web (O’Reilly) has introduced the concept of CSS Post-Processors.

As opposed to the distinct syntax of pre-processors, post-processors typically feed on actual CSS. They can act like polyfills, letting you write to-spec CSS that will work someday and transforming it into something that will work in browsers today. Ideal CSS in, real-life CSS out. Front-end developer already use this concept of bullet-proofing css without even being aware of it. The most popular tool autoprefixer is a post-processor – which takes css and add some vendor prefixes to make it work in as many browsers as possible.

Unlike pre-processors, post-processors are used in modular chunks and can cover-up for the loop holes css pre-processors. However, every technology has its limitation. For example, modular post-processing plugins require a lot of sacrifices. We do not need any more nesting. Instead of an endless number of mixin, we may be bound to CSS specs, like calc or CSS variables. One promising framework for rolling out post-processing plugins is postcss, but it’s too young its documentation is in an awkwardly immature phase.

Conclusively, it's very important to make an informed judgement before choosing a css style for the application we are planning to develop. Rather than following a general rule of thumb to always go for the latest technologies in the market, it is highly important to understand the limitations and underlying factors of our business requirements. CSS3 pre-processors and post-processors can provide very aesthetic look and feel to the user interface of our website. However, we cannot directly make inject it into our project without understanding the real need and benefits of these immerging technologies in our project.

## **References:**

- [1] Lyza Danger Gardner, <http://alistapart.com/column/what-will-save-us-from-the-dark-side-of-pre-processors>, 2015
- [2] Felicity Evans, <http://alistapart.com/article/a-vision-for-our-sass>, 2015
- [3] Miller H. Borges Medeiros, <http://blog.millermedeiros.com/the-problem-with-css-pre-processors/>, 2010
- [4] Chris Loos, <https://www.urbaninsight.com/2012/04/12/ten-reasons-you-should-be-using-css-preprocessor>, 2012
- [5] CHRIS COYIER <https://css-tricks.com/when-using-important-is-the-right-choice/>, 2011
- [6] Blake Simpson, <http://blog.blakesimpson.co.uk/read/37-less-sass-the-advantages-of-css-preprocessing-explained>, 2013
- [7] <http://www.1stwebdesigner.com/power-sass-why-use-css-preprocessors/>
- [8] <http://mobisocial.stanford.edu/papers/boneh-www2012.pdf>, 2012
- [9] Adam Silver, <http://adamsilver.io/articles/the-disadvantages-of-css-preprocessors/>
- [10] Bilal Cinarli, <http://htmlmag.com/article/an-introduction-to-css-preprocessors-sass-less-stylus>, 2014
- [11] Johnathan Croom, <http://code.tutsplus.com/tutorials/sass-vs-less-vs-stylus-preprocessor-shootout--net-24320>, 2012
- [12] <http://www.headfirstlabs.com/books/hf-mw/>
- [13] <http://verekia.com/less-css/dont-read-less-css-tutorial-highly-addictive>
- [14] <http://marketblog.envato.com/resources/10-css-preprocessors-worth-considering/> 2014